ACTon

Documentation

۱. مقدمه

زبان acton یک زبان actor-based است.

زبان‌های actor-based برای مدلسازی، فهم و استدلال درباره ی سیستم های هم‌روند[1] استفاده می‌شوند.

اکتورها واحدهای محاسباتی هم‌روند هستند که قابلیت ارسال و دریافت پیام و واکنش نشان دادن به پیام های دریافتی را دارند. تعامل اکتورها با یکدیگر توسط فرستادن پیام‌های آسنکرون است. هر اکتور یک صندوق پیام[2] دارد که مخصوص ذخیره ی پیام های دریافتی‌اش است.

یک اکتور می‌توانند در پاسخ به پیام‌هایی که دریافت می‌کند:

- به اکتورهای دیگر پیام بفرستد
- اکتورهای جدید بسازد
- متغیرهایش را به روز کند

(در acton تنها قابلیت اول و سوم وجود دارد - یک اکتور نمیتواند اکتور جدید بسازد)

هر اکتور می‌تواند با اکتور هایی در ارتباط باشد که آدرس آن‌ها را در اختیار دارد یا به عبارتی آن‌ها را می‌شناسد.

۲. ساختار کلی

در این زبان، کد برنامه درون یک فایل با پسوند act. قرار دارد.

یک برنامه به زبان acton از قسمت های زیر تشکیل شده است:

- یک یا چند اکتور
  - متغیرهای اکتور[3] (معادل فیلدهای کلاس در زبان‌های شیءگرا)
  - اکتور های شناخته شده برای این اکتور[4]
  - یک initializer در صورت وجود (معادل constructor در زبان‌های شیءگرا)
  - مدیریت کننده های پیام[5] (معادل متدهای کلاس در زبان‌های شیءگرا)
- main

اکتورهای برنامه ساخته می‌شوند و شروع به فعالیت می‌کنند.

---

[1] Concurrent Systems
[2] Mailbox
[3] ActorVars
[4] KnownActors
[5] MsgHandler

۲-۱. قواعد کلی نحو:

زبان acton به بزرگ و کوچک بودن حروف حساس است. در این زبان، وجود کاراکتر های tab و space تاثیری در خروجی برنامه ندارند. جزئیات مربوط به scope و خطوط برنامه در ادامه توضیح داده خواهد شد.

۲-۲. کامنت‌ها:

در این زبان کامنت ها تک خطی هستند و تمامی کاراکتر های بعد از // تا انتهای خط کامنت به حساب می‌آیند.

۲-۳. قواعد نام‌گذاری actor ها، msghandler ها و متغیرها:

اسامی انتخابی برای نام‌گذاری باید از قواعد زیر پیروی کنند:

- تنها از کاراکتر های a..z، A..Z، _ و ارقام تشکیل شده باشند. (محدودیتی روی تعداد کاراکترهای یک اسم در زبان acton وجود ندارد)
- با رقم شروع نشوند.
- معادل کلیدواژه ها نباشند. در جدول زیر تمامی کلیدواژه های زبان acton آمده است:

actor	knownactors	actorvars	extends	initial	msghandler
self	sender	if	else	for	print
false	true	int	boolean	string	main
				break	continue

- نام هر actor یکتاست.
- نام هر msghandler در هر actor یکتاست. به عبارتی msghandler overloading در این زبان وجود ندارد.
- نام هر متغیر در هر scope یکتاست ولی در scope های درونی تر از نام های متغیر بیرونی می‌توان استفاده کرد. در نتیجه در طول آن scope متغیر درونی هنگام استفاده ارجحیت دارد. همچنین overloading برای actorvar ها[۶] در این زبان وجود ندارد.
- نام نمونه اکتورهای ساخته شده در main یکتاست.

---

[۶] امکان تعریف actorvar همنام در یک اکتور و اکتوری که از آن به ارث میبرد.

۳. اکتورها

همانطور که بالاتر به آن اشاره شد، در تعریف هر actor، بخش های ثابتی وجود دارند.

در ابتدا نام actor، نام actor ای که از آن ارث بری می‌کند(در صورت وجود) و queue size (mailbox size) آن مشخص می‌گردد. سپس scope این اکتور توسط {} مشخص می‌شود.

```
actor Child extends Parent (5) {

    // actor body

}
```

سایز queue هر اکتور تنها می‌تواند یک عدد ثابت بزرگتر از 0 باشد. در واقع هر actor یک صف دارد که سایر actor ها پیام های خود را با فراخوانی msghandler های آن actor در آن صف قرار می‌دهند . این صف به طور ضمنی تعریف شده و نیاز به تعریف مستقیم آن نیست. تنها باید طول آن در پرانتز مشخص گردد.

۳-۱. ارث‌بری[۷]

در صورت ارث بری یک actor از یک actor دیگر، تمامی knownactors، actorvars و msghandler های پدر به فرزند به ارث می‌رسد. اما صف هر actor شخصی است و به ارث نمی‌رسد و تنها همان actor به آن دسترسی دارد. همانطور که بالاتر اشاره شد، امکان وجود actorvar های هم‌نام در پدر و فرزند وجود ندارد (actorvar overloading). همینطور امکان وجود دو knownactor با نام یکسان در پدر و فرزند وجود ندارد.

۳-۲. اکتورهای شناخته شده

بخش knownactors مشخص کننده ی actor هایی است که actor فعلی می‌تواند به آن‌ها پیام بفرستد یا به عبارت دقیق‌تر، msghandler آن‌ها را فراخوانی کند. این بخش همواره در اکتورها وجود دارد (حتی اگر اکتور، اکتور شناخته شده نداشته باشد) و لزوما در ابتدای تعریف یک اکتور، با کلیدواژه knownactors مشخص می‌شود.

[7] Inheritance

تعریف هر اکتور شناخته شده به صورت زیر در یک خط جداگانه مشابه declare کردن variable ها انجام می‌شود. ابتدا نوع اکتور و سپس نام اکتور مشخص می‌شود.

```
knownactors {
    Child first;
    Child second;
}
```

۳-۳. متغیرهای اکتور

بخش actorvars مشخص کننده ی متغیر های هر actor است که در تمامی scope آن actor میتوان به آن‌ها دسترسی داشت (همانند فیلد در کلاس). این متغیر ها می‌توانند از جنس int،boolean ،string و آرایه ای از int باشند.

این بخش نیز همواره در اکتورها وجود دارد و پس از بخش knownactors، با کلیدواژه‌ی actorvars مشخص می‌شود. ابتدا تایپ و سپس اسم هر متغیر به صورت زیر در یک خط جداگانه مشخص می‌شود.

```
actorvars {
    string name;
    int numOfSiblings;
    boolean isOnlyChild;
}
```

۴. msghandler ها

در هر msghandler، پیاده سازی عملکرد مربوط به پیام متناظر با آن تعریف می‌شود. در واقع هر actor مداوما از صف خود به ترتیب یک msg برمی‌دارد و بسته به نوع msg، handler مربوطه‌ی آن را اجرا می‌کند و پس از اتمام اجرا، به سراغ msg بعدی در صف می‌رود. در حین اجرای یک msghandler از یک actor، سایر msghandler های این actor نمی‌توانند اجرا شوند.

مسج هندلرها با کلیدواژه‌ی msghandler و پس از آن نام هندلر مشخص می‌شوند. پس از نام msghandler، مشابه تعریف متد در جاوا، داخل پرانتز آرگومان‌های ورودی msghandler مشخص می‌شود که می‌توانند از جنس int، boolean، string و یا []int باشند. scope هندلر با {} تعیین می‌شود.

```
msghandler sendBackTimesN(string text, int n) {
    for(; n > 0; n = n - 1)
        sender.send(text);
}
```

تعریف msghandler در acton از این نظر با جاوا تفاوت دارد که مقدار بازگشتی برای آن وجود ندارد.

همچنین در این زبان قابلیت overriding برای msghandler ها وجود ندارد.

۴-۱. initializer

اولین msghandler یک actor در صورت وجود، می‌تواند initializer آن باشد که با کلیدواژه ی initial به عنوان نام آن مشخص می‌گردد و همانند سایر msghandler ها تعریف می‌شود. Initializer در واقع معادل همان constructor است که در هنگام ساخت یک اکتور جدید فراخوانی می‌شود.

```
msghandler initial() {
    self.siblings = 2;
    self.isOnlyChild = false;
}
```

۴-۲. تعریف متغیرهای محلی

تمامی متغیرهای محلی فقط در ابتدای msghandler تعریف می‌شوند و در ابتدای هیچ scope جدید دیگری قابلیت تعریف متغیر محلی وجود ندارد. تعریف متغیرهای محلی مشابه تعریف actorvar های یک اکتور می‌باشد؛ ابتدا تایپ و سپس اسم هر متغیر در یک خط جداگانه مشخص می‌شود.

۴-۳. فراخوانی

فراخوانی یک msghandler (ارسال پیام) در این زبان یک statement است چرا که مقدار بازگشتی ندارد. یک اکتور می‌تواند به mailbox خودش، mailbox یکی از knownactor هایش یا mailbox فرستنده ی پیامی که دریافت می‌کند، پیام بفرستد:

```
self.foo();
sender.foo();
someKnownActor.foo();
```

۵. main

پس از تعریف[8] اکتور ها، در انتهای برنامه نمونهٔ[9] اکتورهای موجود در برنامه در بخش نهایی main ساخته شده و شروع به فعالیت می‌کنند. در این بخش فقط تعداد متناهی نمونه از اکتور های تعریف شده ساخته می‌شود. برای هر نمونه از اکتور، binding آن به knownactor هایش در لیست اول ورودی‌ها و آرگومان‌های initializer آن در صورت وجود در لیست دوم ورودی‌ها داده می‌شود. این دو لیست ورودی توسط : از هم جدا می‌شوند. نمونه ای از کد این بخش در زیر آمده است:

```
main {
    A a1(b):();
    A a2(b):();
    B b(a1, a2):(0);
}
```

۶. کلیدواژه‌ی self:

این کلیدواژه همانند همتای خود در سایر زبان‌ها، به actor ای که در آن قرار داریم اشاره می‌کند. در این زبان از self برای فراخوانی msghandler های actor و همچنین برای دسترسی به actorvar های آن استفاده می‌گردد. اما کلیدواژه ی self در این زبان به تنهایی نمی‌تواند استفاده شود.

دقت شود که دسترسی به actorvar بدون این کلیدواژه نیز امکان پذیر است. در صورت وجود actorvar و متغیر محلی با یک نام، در صورت عدم استفاده از این کلیدواژه، متغیر محلی منظور است. (با توجه به اینکه در scope درونی تعریف شده است)

۷. کلیدواژه‌ی sender:

گیرنده ی پیام، می‌تواند به فرستنده ی آن با استفاده از این کلیدواژه reference ای داشته باشد. امکان فراخوانی msghandler روی این کلیدواژه وجود دارد ولی منطقا کلاس فرستنده باید آن msghandler را داشته باشد.

---

[8] definition
[9] instance

دقت کنید امکان دسترسی به actorvar های اکتور دیگر از طریق این کلیدواژه یا به هر نحو دیگر امکان پذیر نیست.

۸. انواع داده ها:

تعریف متغیر در این زبان مانند زیر انجام می‌شود:

```
string name;
boolean isFemale;
int studentID;
int gpa;
int grades[10];
```

در این زبان امکان تعریف چند متغیر در یک خط وجود ندارد.

همچین امکان مقداردهی متغیرها در هنگام تعریف (در همان خط) نیز وجود ندارد.

در صورتی که به متغیر های از جنس int، boolean و string مقدار اولیه نسبت داده نشود، مقدار اولیه آن‌ها برابر با مقدار پیش‌فرض تایپ خود در نظر گرفته می‌شود. مقادیر پیش فرض آن‌ها به صورت جدول زیر است:

int	0
boolean	false
string	“”

در صورتی که متغیر از جنس آرایه باشد، مقدار اولیه پیش‌فرض تمام خانه های آن برابر با 0 است.

تایپ های موجود در این زبان به صورت زیر است:

int
boolean
string
int[]

سه تایپ اول از نوع primitive هستند (خود مقادیر در آن‌ها ذخیره می‌شوند نه پوینتری به خانه ای از حافظه) و تایپ آخر از نوع non-primitive است و در آن پوینتری به خانه ای از حافظه وجود دارد.

برای تعریف متغیر از نوع آرایه، مقداردهی آن و همچنین دسترسی به آن به شکل زیر عمل می‌کنیم:

```
int a[1];
a[0] = 2;
b = a[2];
```

لازم به ذکر است که اندازه ی یک آرایه تنها می‌تواند یک عدد ثابت بزرگتر از صفر باشد. (دقت کنید عبارتی مثل 2+3 نیز که مقدار ثابت دارد، به عنوان اندازه‌ی آرایه قابل قبول نیست)

۸. عملگرها

عملگرها در زبان acton به چهار دسته ی عملگرهای حسابی، مقایسه ای، منطقی و تخصیص تقسیم می‌شوند.

۸-۱.عملگرهای حسابی

این دسته از عملگرها تنها روی اعداد عمل می‌کنند. لیست این عملگر ها در جدول زیر آمده است. در مثال های استفاده شده A را برابر با ۲۰ و B را برابر با ۱۰ درنظر بگیرید.

عملگر	شرکت‌پذیری	توضیح	مثال
+	چپ	جمع	A + B = 30
-	چپ	تفریق	A – B = 10
*	چپ	ضرب	A * B = 200
/	چپ	تقسیم	A/ B = 2 B / A = 0
%	چپ	باقی‌مانده	A % B = 10
-	راست	منفی تک عمل‌وندی	-A = -20
++ و --	راست	پیشوندی	--A
++ و --	چپ	پسوندی	A++

۸-۲. عملگرهای مقایسه ای

این عملگرها وظیفه ی مقایسه را دارند؛ پس نتیجه ی آن‌ها باید مقدار true یا false باشد. یعنی خروجی آن‌ها یک boolean است.

توجه داشته باشید که عملوندهای عملگرهای > و < تنها از جنس اعداد صحیح هستند.

همچنین برای عملگر == و != نیز باید تایپ عملوندها یکسان باشند و در صورت آرایه بودن، اندازه ی آن‌ها نیز برابر باشد. در غیر این صورت باید خطای کامپایل گرفته شود.

لیست عملگرهای مقایسه ای در جدول زیر آمده است. مقادیر A و B را همانند قبل درنظر بگیرید.

عملگر	شرکت‌پذیری	توضیح	مثال
==	چپ	تساوی	(A == B) = false
!=	چپ	عدم تساوی	(A != B) = true
<	چپ	کوچکتر	(A < B) = false
>	چپ	بزرگتر	(A > B) = true

۸-۳. عملگرهای منطقی

در این زبان، عملگرهای منطقی تنها روی تایپ Boolean قابل اعمال است. این عملگرها در جدول زیر آمده است. A را برابر true و B را برابر false درنظر بگیرید.

عملگر	شرکت پذیری	توضیح	مثال
&&	چپ	عطف منطقی	(A && B) = false
||	چپ	فصل منطقی	(A || B) = true
!	راست	نقیض منطقی	(!A) = false

۸-۴. عملگر شرطی سه تایی

این عملگر دارای سه عملوند است که تایپ عملوند اول Boolean است و تایپ دو عملوند بعدی باید یکسان باشد. در صورت true بودن عملوند اول، مقدار عملوند دوم و درغیر این صورت مقدار عملوند سوم را برگردانده می‌شود.

دقت شود که عبارتی که جای عملوندهای دوم قرار میگیرد طوری پارس میشود که انگار پرانتزگذاری شده است. یعنی عبارت، فارغ از اولویت عملگرهای داخل آن به گونه ای ارزیابی می‌شود که اگر داخل پرانتز بود به آن صورت ارزیابی می‌شد.

همچنین شرکت‌پذیری این عملگر به صورت راست به چپ است. یعنی عبارت a?b:c?d:e به صورت a?b:(c?d:e) ارزیابی می‌شود.

۸-۵. عملگر تخصیص

این عملگر که به صورت = نمایش داده می‌شود وظیفه ی تخصیص را برعهده دارد. یعنی مقدار عملوند سمت راست را به عملوند سمت چپ اختصاص میدهد. برای آرایه مقدار تک تک عناصر آرایه ی سمت راست را به عناصر آرایه ی سمت چپ تخصیص میدهد. دقت شود که برای استفاده از این عملگر برای دو آرایه، اندازه ی آرایه ها باید برابر باشد.

همچنین دقت داشته باشید که عملوند سمت چپ باید از نوع lvalue باشد. مفهوم lvalue و rvalue در زبان acton مشابه زبان C است. عبارات lvalue عباراتی هستند که به یک مکان در حافظه اشاره می‌کنند. در مقابل عبارات rvalue به مکان خاصی در حافظه اشاره نمی‌کنند و صرفا یک عبارت دارای مقدار هستند. به عنوان مثال یک متغیر یا یک دسترسی به یکی از عناصر آرایه یک عبارت lvalue است اما عبارت ۳ + ۱۰ یک عبارت rvalue محسوب می‌شود. عبارات rvalue تنها در سمت راست عملگر تخصیص قرار می‌گیرند.

۸-۶. اولویت عملگرها

اولویت عملگرها طبق جدول زیر است:

اولویت	دسته	عملگرها	شرکت‌پذیری
1	پرانتز	()	چپ به راست
2	دسترسی به عناصر آرایه تک عملوندی پسوندی	[] ++ --	چپ به راست
3	تک عملوندی پیشوندی	-- ++! -	راست به چپ

4	ضرب و تقسیم و باقی مانده	* /٪	چپ به راست
5	جمع و تفریق	+ -	چپ به راست
6	رابطه ای	< >	چپ به راست
7	مقایسه ی تساوی	== =!	چپ به راست
8	عطف منطقی	&&	چپ به راست
9	فصل منطقی	||	چپ به راست
10	شرطی سه تایی	? :	راست به چپ
11	تخصیص	=	راست به چپ
12	کاما(ورودی مسج هندلرها)	,	چپ به راست

۹. گزاره‌ها

۹-۱. ساختار تصمیم‌گیری

در زبان acton تنها ساختار تصمیم‌گیری، if...else است. ساختار نحوی[1] آن مشابه زیر است:

```
if(num[i] > max){
    max = num[i];
    max_idx = i;
}
else
    if(num[i] < min){
        min = num[i];
        min_idx = i;
    }
```

این ساختار بدون else نیز می‌تواند بکار رود.

[1] Syntax

۹-۲. ساختار تکرار

تنها ساختار تکرار در این زبان for است. ساختار نحوی آن به این صورت است که ۳ عبارت به عنوان مقداردهی، شرط و به روزرسانی می‌گیرد که عبارت مقداردهی و به روزرسانی آن یک تساوی است و عبارت شرط هرعبارتی از تایپ boolean می‌تواند باشد. هرکدام از عبارات مقداردهی، شرط و بروزرسانی for میتوانند خالی باشند.

داخل ساختار تکرار می‌توان از statement های break و continue استفاده کرد.

دقت شود که تمامی متغیرها در ابتدای برنامه تعریف می‌شوند و متغیری را در ۳ عبارت بالا یا در داخل اسکوپِ for تعریف نمی‌کنیم.

۱۰. scope ها

۱۰-۱. scope های موجود در زبان

به طور کلی در زبان acton موارد زیر در scope جدیدی قرار دارند:

- خطوط کد داخل یک actor
- پارامتر ها و خطوط کد داخل یک msghandler
- داخل گزاره‌های for و if
- داخل scope های جدیدی که با {} مشخص می‌شوند

۱۰-۲. قوانین scope ها

- تعریف actor ها در بیرونی ترین scope است.
- متغیرهایی که در داخل یک scope تعریف می‌شوند در scope های بیرون آن دسترس‌پذیر نیستند و صرفا در scope های درون آن قابل دسترس هستند.
- امکان تعریف متغیر با نام یکسان در یک scope وجود ندارد اما در scope های درونی آن امکان تعریف مجدد وجود دارد و تا زمان خروج از scope درونی، نزدیک ترین تعریف به آن استفاده می‌شود. مگر اینکه از کلیدواژه ی self استفاده شود.
- داخل msghandler ها، statement ها می‌توانند داخل تعداد دلخواهی {} اضافی و تو در تو (scope های جدید) قرار بگیرند.

۱۰-۳. قوانین خطوط برنامه

قوانین خطوط این زبان مشابه زبان Java می‌باشد تمامی دستورات، در انتهای خود یک کاراکتر ; دارند.

۱۱. توابع پیش‌فرض

در زبان acton، تنها یک تابع پیش‌فرض وجود دارد و آن print است.

این تابع به صورت ضمنی تعریف شده است و می‌تواند یک آرایه یا یک مقدار int یا string یا boolean را دریافت کند و مقدار آن را در کنسول چاپ کند؛ یعنی آرگومان ورودی آن یک عبارت است.

```
print("hello world!");
```

۱۲. مثال ها

در ادامه چندین کد نمونه از این زبان آمده است:

۱.

```
actor Calculator(5) {
    knownactors {
        Taylor t;
    }

    actorvars {
        int latest_fact;
    }

    msghandler initial() {
        latest_fact = 0;
    }

    msghandler factorial(int n) {
        int temp;
        temp = 1;
        for(;;) {
            if(n == 0)
                break;
            temp = temp * n;
            n = n - 1;
        }
        latest_fact = temp;
    }

    msghandler get_fact(int n) {
        self.factorial(n);
        if(sender == t)
            sender.pass(self.latest_fact);
        else
            print("can't send msg to unknown actor");
    }

}

// definition of Taylor also goes here

main {
    Calculator c(t):();
    Taylor t(c):();
}
```

```
actor BridgeController(5) {

    knownactors {
        Train t1;
        Train t2;
    }

    actorvars {
        boolean isWaiting1;
        boolean isWaiting2;
        boolean signal1;
        boolean signal2;
    }

    msghandler initial() {
        signal1 = false;
        signal2 = false;
        isWaiting1 = false;
        isWaiting2 = false;
    }

    msghandler Arrive() {
        if (sender == t1){
            if (signal2 == false) {
                signal1 = true;
                t1.YouMayPass();
            }
            else {
                isWaiting1 = true;
            }
        }
        else {
            if (signal1 == false){
                signal2 = true;
                t2.YouMayPass();
            }
            else{
                isWaiting2 = true;
            }
        }
    }

    msghandler Leave() {
        if (sender == t1) {
            signal1 = false;
            if (isWaiting2){
                signal2 = true;
                t2.YouMayPass();
                isWaiting2 = false;
            }
        } else {
            signal2 = false;
            if (isWaiting1) {
                signal1 = true;
                t1.YouMayPass();
                isWaiting1 = false;
            }
        }
    }
}
```

```
actor Train(3) {
    knownactors {
        BridgeController controller;
    }

    actorvars {
        boolean onTheBridge;
    }

    msghandler initial() {
        onTheBridge = false;
        self.Passed();
    }

    msghandler YouMayPass() {
        onTheBridge = true;
        print("You may pass!");
        self.Passed();
    }

    msghandler Passed() {
        onTheBridge = false;
        controller.Leave();
        self.ReachBridge();
    }

    msghandler ReachBridge() {
        controller.Arrive();
    }
}
```

```
main {
    Train train1(theController):();
    Train train2(theController):();
    BridgeController theController(train1, train2):();
}
```